A Tool Suite for the Automated Synthesis of Security Function Chains

Nicolas Schnepf, Rémi Badonnel, Abdelkader Lahmadi, Stephan Merz

▶ **To cite this version:**

Nicolas Schnepf, Rémi Badonnel, Abdelkader Lahmadi, Stephan Merz. A Tool Suite for the Automated Synthesis of Security Function Chains. IFIP/IEEE IM 2019 - IFIP/IEEE International Symposium on Integrated Network Management, Apr 2019, Washington, United States. hal-02111658

HAL Id: hal-02111658
https://inria.hal.science/hal-02111658

Submitted on 26 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Tool Suite for the Automated Synthesis of Security Function Chains

Nicolas Schnepf, Rémi Badonnel, Abdelkader Lahmadi, and Stephan Merz
Université de Lorraine, CNRS, INRIA, LORIA, F-54000 Nancy, France
{schnepf, badonnel, lahmadi, merz}@inria.fr

Abstract

Software-defined networking may serve as a support for the elaboration of security chains capable of protecting end-user devices. These chains may be composed of different security functions, such as firewalls and intrusion detection systems. This demonstration showcases a tool suite for automating such a generation, from the learning of the behavior of applications, to the factoring and instanciation of security chains.

I. Background

The programmability that characterizes software-defined networking (SDN) [1] simplifies the definition and enforcement of network policies by decoupling the control and the data planes. In particular, end users can be protected by means of chains of security functions, as described in [2]. These chains are built up by combining (in sequence or in parallel) elementary security functions such as intrusion detection systems, firewalls or data leakage prevention mechanisms. The chains can conveniently be described using high level programming languages such as Pyretic [3]. This language, part of the Frenetic family of languages [4], is embedded in Python and provides support for compiling the specified policies into low-level OpenFlow rules.

However, the complexity of such security chains induces the risk of introducing misconfigurations and security holes in the network. Several authors proposed the use of formal methods [5], [6], and Pyretic also supports such methods through its extension Kinetic [7] that integrates model checking support for the verification of the control plane. Unfortunately, applying such verification mechanisms at runtime is prohibitive because of the excessive execution time they require. We therefore propose a framework for the automated synthesis of chains based on specifications of generic network policies and the observed behavior of applications. The chains generated in this way are guaranteed to satisfy certain properties such as shadowing freedom and coherence with the network policies, as well as the absence of routing cycles and black holes. We developed the framework with the objective of protecting Android applications, although it could in principle be instantiated for other applicative contexts.

II. The Synaptic Platform

Our system, called Synaptic, mainly relies on an orchestrator that integrates several functionalities for automating the generation of security chains, as illustrated in Fig. 2. It first supports process learning for building Markov models of the applications to be protected [2]. These models are built by aggregating application flows collected through a network probe [8]. States of our Markov models represent organizations that own IP addresses contacted by the application, and state transitions are obtained by computing the probability of moving from one state to another. Figure 1 presents a simple automaton computed in this way, each state corresponding to a collection of network flows to destinations owned by the organization labelling the state.

These automata, together with generic specifications of network policies provided by the network administrator and the set of permissions requested by the application, serve as an input for classifying flows as either safe or representing potential attacks such as denial of service, port scanning, worms or botnets [10]. The classification is performed through declarative logic programming rules and results in predicates in first-order logic that identify potential attacks. From there, another set of rules generates a high-level representation of a chain of security functions specific to the considered application that is finally compiled to a Pyretic program.

Although the chains obtained in this way could in principle be deployed in an SDN environment, there may be redundancies between chains generated for different applications. The last step of our orchestration process, illustrated in Fig. 3, therefore consists in factorizing the chains generated for different applications. The combination of chains of security functions before their deployment is non-trivial. Rather than simply composing the different chains in parallel, our factorization algorithm [9] identifies and integrates common security

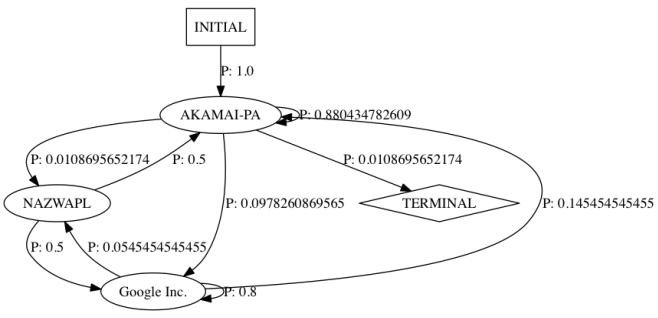

Figure 1. Automaton describing the behavior of an Android application.

978-3-903176-15-7 © 2019 IFIP

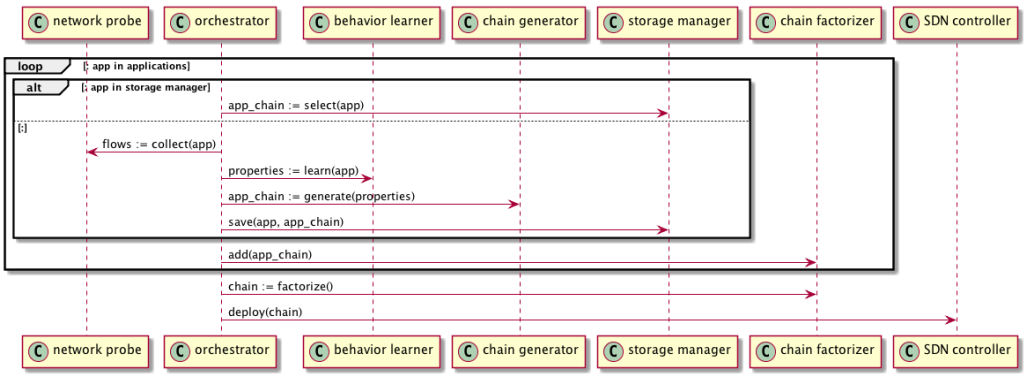

Figure 2. Proposed system for generating and merging security chains.

functions across applications. It relies on the fixed order in which security functions appear in the chains generated by our synthesis module for preserving correctness. Experimental evaluation showed the potential for significant reductions in the number of security functions to be deployed after factorization, while preserving the accuracy of the chains.

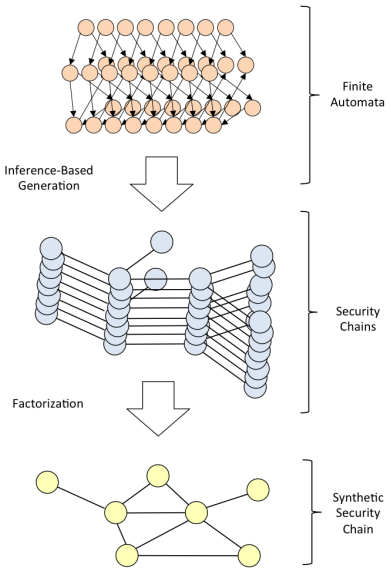

Figure 3. Overall process for generating and merging security chains.

III. The Demonstration

In this demo we will present the whole process of security chain generation, from learning the behavior of applications to the factorization of the corresponding chains. We will illustrate this process with several applications that can be either safe or contain malicious traffic that must be blocked. We will show how these different security chains can be combined in practice to build a larger security chain protecting every input application provided to our orchestrator and how such a chain can be compiled and deployed in the network.

We will also demonstrate how we integrate logic programming in our overall orchestration process and how this logic based inference model can be used with different detection methods by decoupling the specification of the chain from the representation of security requirements of end users. Finally, we will give some examples of such requirements, specified as first order predicates and how they are used to generate security chains. We will present different approaches for the generation of chains of security functions, explain their advantages and limits and illustrate the effectiveness and efficiency of our factorization algorithm for combining chains.

References

[1] N. Feamster and H. Kim, "Software-Defined Networks: Improving Network Management with SDN," in *IEEE Communications Magazine*, February 2013.

[2] N. Schnepf, S. Merz, R. Badonnel, and A. Lahmadi, "Towards Generation of SDN Policies for Protecting Android Environments based on Automata Learning," in *Proceedings of the 16th Network Operations and Management Symposium (IEEE/IFIP NOMS'18)*, 2018.

[3] N. Foster, M. J. Freedman, A. Guha, R. Harrison, N. P. Kata, C. Monsanto, J. Reich, M. Reitblatt, R. Jennifer, C. Schlesinger, A. Story, and D. Walker, "Languages for Software-Defined Networks," in *Software Technology Group*, 2016.

[4] N. Foster, M. J. Freedman, R. Harrison, C. Monsanto, and D. Walker, "Frenetic, a Network Programming Language," in *Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming (ICFP'11)*, 2011.

[5] E. Al-Shaer and S. Al-Haj, "FlowChecker, Configuration Analysis and Verification of Federated OpenFlow Infrastructures," in *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration (CCS'10)*, 2010.

[6] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky, "Vericon: Towards Verifying Controller Programs in Software-Defined Networks," in *Proc. 35th ACM SIGPLAN Intl. Conf. Programming Language Design (PLDI'14)*, Edinburgh, UK, 2014, pp. 282–293.

[7] H. Kim, J. Reich, A. Gupta, M. Shahbaz, N. Feamster, and R. Clark, "Kinetic: Verifiable Dynamic Network Control," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15)*, 2015.

[8] A. Lahmadi, F. Beck, E. Finickel, and O. Festor, "A platform for the analysis and visualization of network flow data of android environments," IFIP/IEEE International Symposium on Integrated Network Management (IM), May 2015, poster.

[9] N. Schnepf, S. Merz, R. Badonnel, and A. Lahmadi, "Automated factorization of security chains in software-defined networks," in *Proceedings of the 16th IFIP/IEEE Symposium on Integrated Network and Service Management (IM 2019)*, 2019.

[10] ——, "Rule-Based Synthesis of Chains of Security Functions for Software-Defined Networks," in *Proceedings of the 18th International Workshop on Automated Verification of Critical Systems (AVOCS'18)*, 2018.